Attorney's Do[illegible]No.: 10559-610001 / P12849

APPLICATION

FOR

UNITED STATES LETTERS PATENT

10041578.010702

TITLE:	QUEUE ARRAY CACHING IN NETWORK DEVICES
APPLICANT:	GILBERT WOLRICH, MARK B. ROSENBLUTH AND DEBRA BERNSTEIN

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. ET679041717US

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

January 7, 2002
Date of Deposit

[illegible]
Signature

Gabe Lewis
Typed or Printed Name of Person Signing Certificate

QUEUE ARRAY CACHING IN NETWORK DEVICES

BACKGROUND

This invention relates to queue arrays for use in network devices.

Network devices such as routers and switches can have line speeds that can be faster than 10 Gigabits. For maximum efficiency the network device should be able to process data packets, storing them to and retrieving them from memory at a rate at least equal to the line rate. However, current network devices may lack the necessary speed to process data packets at the line speeds.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a network system.

FIG. 2 is a block diagram of a network device.

FIG. 3 shows a queue and queue descriptor.

FIG. 4 illustrates an enqueue and a dequeue operation.

DETAILED DESCRIPTION

Referring to FIG. 1, a network system 2 for processing data packets includes one or more sources 4 of data packets coupled to a network device 6 and one or more destinations 8 for the data packets. Each source 4 can include other network devices connected over a communications path operating at high data packet transfer line speeds. Examples of such communications paths include an optical carrier (OC)-192 line, and a 10-Gigabit line. Likewise, the destinations 8 also can include other network devices, as well as a similar network connection.

The network device 6 includes a processor 10 that uses a memory (not shown) storing memory data structures. The processor executes instructions and operates with the memory data structures as configured to receive, store and forward the data packets to a specified destination. The network device 6 can be part of, a network switch or a network router and so forth. The processor 10 also includes one or more programming engines. The programming engine ("PE") includes a sixteen-entry content addressable memory ("CAM"). The CAM tracks, which of its entries is the least-recently-used ("LRU").

Referring to FIG. 2, the network device 6 includes memory 14 coupled to the processor 10. The memory 14 stores output queues 18 and their corresponding queue descriptors 20. The processor 10 includes memory controller logic 38 that includes a cache 12 to store some of the queue descriptors 20 as described below. The processor 10 also has a queue manager 42 that can be implemented as a programming engine. A CAM 44 serves as a tag store holding the addresses of queue descriptors 20 that are stored in the cache.

The queue manager 42 receives enqueue requests from a set of programming engines that function as a receive pipeline 46. The receive pipeline 46 is programmed to process and classify data packets received by the network device 6 from sources 4 (FIG. 1). The enqueue requests specify which output queue 18 an arriving packet should be added to. Another programming engine functions as a transmit scheduler 48 to send dequeue requests to the queue manager 42. The dequeue requests specify the output queue 18 from which a packet is to be removed for transmittal to a destination 8 (FIG. 1).

An enqueue operation adds information that arrived in a data packet to one of the output queues 18 and updates the corresponding queue descriptor 20. A dequeue operation removes information from one of the output queues 18 and updates the corresponding queue descriptor 20, to allow the network device 6 to transmit the information to the appropriate destination 8.

An example of an output queue 18 and its corresponding queue descriptor 20 is shown in FIG. 3. The output queue 18 includes a linked list of elements 22, each of which contains a pointer 24 to the next element 22 in the output queue 18. The pointer 26 of the last element 22 in the queue 18 contains a null value. A function of the address of each element 22 implicitly maps to the information 26 stored in the memory 14 that the element 22 represents. For example, the first element 22a of output queue 18 shown in FIG. 3 is located at address A. The location in memory of the information 26a that element 22a represents is implicit from the element's address A, illustrated by dashed arrow 27a. Element 22a contains the address B, which serves as a pointer 24 to the next element 22b in the output queue 18, located at address B.

The queue descriptor 20 includes a head pointer 28, a tail pointer 30 and a count 32. The head pointer 28 points to the first element 22 of the output queue 18, and the tail pointer 30 points to the last element 22 of the output queue 18. The count 32 identifies the number (N) of elements 22 in the output queue 18.

Executing enqueue and dequeue operations for a large number of queues 18 in the memory 14 at high-bandwidth line rates can be accomplished by storing some of the queue descriptors 20 in the cache 12 (FIG. 2). The queue manager 42 implements a software-controlled tag store in its CAM 44 to identify the addresses in memory 14 of the sixteen queue descriptors 20 most-recently-used in enqueue or dequeue operations. The cache 12 stores the corresponding queue descriptors 20 (the head pointer 28, tail pointer 30 and count 32) stored at the addresses identified in the tag store 44.

The queue manager 42 issues commands to return queue descriptors 20 to memory 14 and fetch new queue descriptors from memory such that the queue descriptors stored in the cache 12 remain coherent with the addresses in the tag store 44. The queue manager 42 also issues commands to the memory controller logic 38 to indicate which queue descriptor 18 in the cache 12 should be used to execute the command. The commands that reference the head pointer 28 or tail pointer 30 (see FIG. 3) of a queue descriptor 20 in the cache 12 are executed in the order in which they arrive at the memory controller 38.

Referring to FIG. 4, when performing an enqueue operation, the address in memory 14 of a new element 22e to be added to the queue 18 is stored (as indicated by dashed line 40) in the pointer 24d of the element 22d that currently is at the address indicated by the tail pointer 30 for that queue. The address of the new element 22e address then is stored in the tail pointer 30 of the corresponding queue descriptor 20 in the cache 12, as indicated by dashed line 31. Because only a single write operation to memory 14 is required for an enqueue operation, only two cycles are required to update the cache 12. Subsequent enqueue operations to the same queue 18 then can be initiated.

For dequeue operations, the address contained in the head pointer 28 is returned to the queue manager 42 (FIG. 2) to indicate (by implicit mapping) the location in memory 14 of the information 26e to be sent to a specified destination device 8 (FIG. 1). The pointer 24a in the element 22a is read to obtain the address of the next element 22b in the queue 18. The address of next element 22b is written to the head pointer of the corresponding queue descriptor 20 in the cache 12 (indicated by dashed line 29). Subsequent dequeue operations to the same queue 18 are delayed until the head pointer 28 in the cache 12 is updated. However, so long as the element 22 being read is not the only element in the queue 18, an enqueue operation with respect to the queue 18 can proceed even if a dequeue operation is in progress because the tail pointer 30 is not affected by the dequeue operation.

An advantage of locating the cache 12 of queue descriptors 20 at the memory controller logic 38 includes allowing for low latency access to and from the cache 12 and the memory 14. Also, having the control structure for queue operations in a programming engine can allow for flexible high performance while using existing micro-engine hardware.

Various features of the system can be implemented in hardware, software or a combination of hardware and software. For example, some aspects of the system can be implemented in computer programs executing on programmable computers. Each program can be implemented in a high level procedural or object-oriented programming language to communicate with a computer system. Furthermore, each such computer program can be stored on a storage medium, such as read only memory (ROM) readable by a general or special purpose programmable computer, for configuring and operating the computer when the storage medium is read by the computer to perform the functions described above.

Other implementations are within the scope of the following claims.